

# Supplementary Information - NeKo: a tool for automatic network construction from prior knowledge

Ruscone et al.

October 14, 2024

This document contains additional details and analysis that complement the work presented in the main text. The code and figures used for both the main article and the supplementary materials can be found at the following github page: [https://github.com/sysbio-curie/NeKo\\_Supp\\_Mat](https://github.com/sysbio-curie/NeKo_Supp_Mat).

## Contents

<b>S1 Package availability and composition</b>	<b>1</b>
<b>S2 Troubleshooting</b>	<b>2</b>
<b>S3 Link between Python functions and NeKo strategies</b>	<b>3</b>
<b>S4 How to integrate a new database in the Neko's Universe class</b>	<b>3</b>
<b>S5 Additional analysis on Use case 1 - differences between Signor and HuRI</b>	<b>5</b>
<b>S6 Additional analysis on Use case 2 - Topological differences between Flobak et al. and NeKo</b>	<b>11</b>

## S1 Package availability and composition

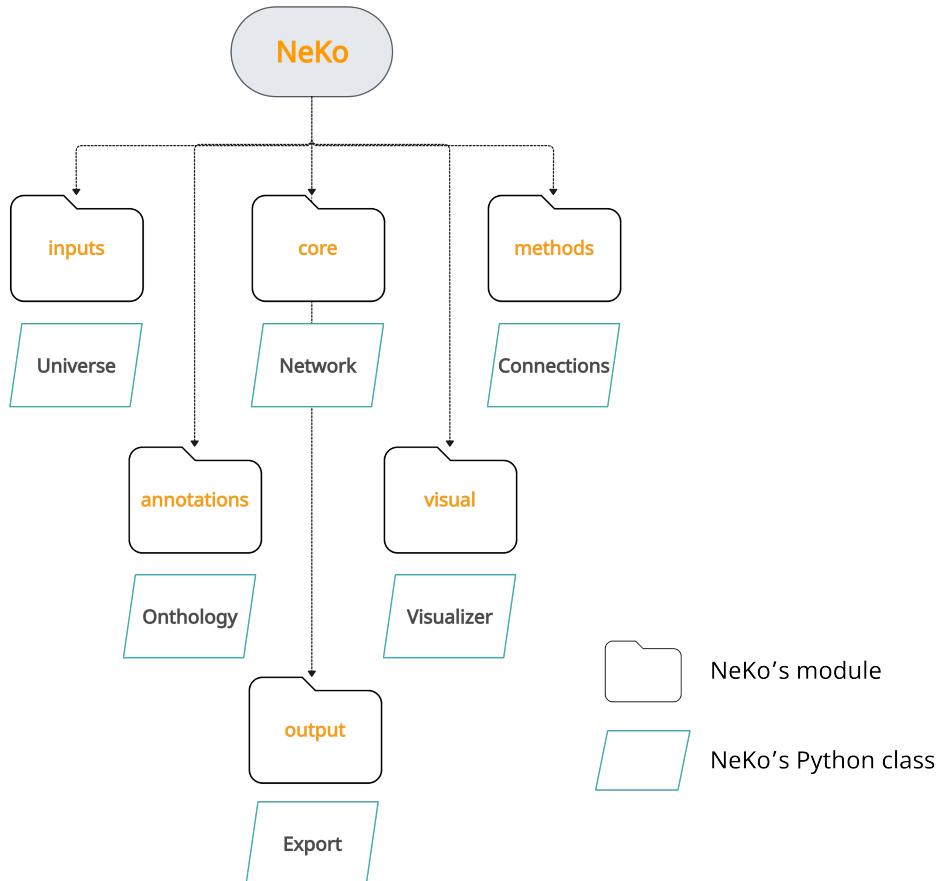
The source code of NeKo is available on GitHub at the following page: <https://github.com/sysbio-curie/Neko>. In addition to the modules represented in figure S1, the package includes a folder "data" with dedicated functions to import Signor (requires a separate download) and the PhosphositePlus databases. NeKo can be installed via:

```
pip install nekomata
```

or directly by source using:

```
git clone https://github.com/sysbio-curie/Neko
cd Neko
pip install .
pip install -r requirements.txt
```

In addition, we included a tutorial made with Jupyter notebooks to showcase NeKo's main features and showing how to use the classes that compose the package. The tutorial is available at the following link: <https://sysbio-curie.github.io/Neko/>. The notebooks are also included in the folder "notebooks" on github (<https://github.com/sysbio-curie/Neko/tree/main/notebooks>) ready to run.



**Figure S1.** Composition of the NeKo package. Each package module (represented by a folder) contains the classes that compose the package. The module "core" contains the `Network` class, while "inputs" contains the `Universe`. The "methods" module contains the `Connections` class. The "annotations", "visual" and "output" contain respectively the `Ontology`, `Visualizer`, and `Export` classes.

## S2 Troubleshooting

Despite the package installation has been extensively tested on multiple platform, it could be possible to encounter some errors during the installation of NeKo. For now, the most common ones are related to Graphviz and Pypath. Upon importing NeKo in a Jupyter Notebook, in particular the module `NetworkVisualizer`, an error regarding the Graphviz installation could arise. In this case, make sure to have pygraphviz installed in your conda/mamba environment (it is a NeKo requirement, so it should be automatically downloaded when installing NeKo). Once verified, make sure Graphviz is installed in your machine, outside the environment. You can install Graphviz on Linux system with the following command:

```
sudo apt install graphviz
```

or on Mac system:

```
brew install graphviz
```

For more details visit: <https://graphviz.org/download/>.

The Pypath package is a component of NeKo that handles the translation of identifiers from Genesymbol to Uniprot. When importing NeKo for the first time, Pypath will create a folder (usually in your home, `/home`) containing some cache files necessary for its usage. During the process of

downloading these files, some errors could occur, preventing Pypath from reading them correctly. This will rise some errors in the execution of the basic NeKo operation. One possible solution is to locate and empty the cache folder, forcing a new download of those files at the next NeKo import.

More information can be found in the Pypath book: <https://pypath.omnipathdb.org/notebooks/manual.html>.

Other errors concerning the NeKo package can be reported in the Issues page of the NeKo GitHub page: <https://github.com/sysbio-curie/Neko/issues>.

## S3 Link between Python functions and NeKo strategies

In the network class the user can find different functions that allows to run the strategies explained in the main text [??](#). Table [??](#) illustrates which function correspond to which strategy.

Each function can be adapted to the user's preferences by adjusting the parameters. We report in the following list a description of the parameters that characterize the function `complete_connection`, which implements the Reciprocal Pathway Extender strategy.

- `Network.complete_connection`
  - `maxlen`: The maximum length of the paths to be searched for. Default is 2.
  - `algorithm`: The search algorithm to be used. It can be '`bfs`' (Breadth-First Search) or '`dfs`' (Depth-First Search).
  - `minimal`: A boolean flag indicating whether to update the network every time a new path is added in the nodes iteration. If set to `False`, it will look for paths between nodes pair in the Universe ignoring existing paths in the network. Default is `True`.
  - `only_signed`: A boolean flag indicating whether to filter unsigned paths. Default is `False`.
  - `consensus`: A boolean flag indicating whether to check for consensus among references. Default is `False`.
  - `connect_with_bias`: A boolean flag that controls whether new nodes are immediately connected to their neighbors when first added to the network. The default is `False`. If set to `True`, each time a new node is introduced via a new path, the function attempts to connect the node to its immediate neighbors that are already in the network. This can save time by reducing the need to search for paths between pairs of nodes later. However, using this option comes with a trade-off. While it improves efficiency by skipping iterative searches, it also risks missing potential longer or alternative paths that might have been discovered through a more exhaustive search. Therefore, the choice depends on whether you prioritize faster computation (with possible path omissions) or a more thorough pathfinding process (which may take longer but captures all possible connections).

The API of NeKo (<https://sysbio-curie.github.io/Neko/api.html>) contains a description of the parameters of all the functions included in the Network class.

## S4 How to integrate a new database in the Neko's Universe class

NeKo by default implements the Omnipath database [\[?\]](#) in the Universe class. Omnipath can be initialized directly from the Network class:

```
neko_network = Network(list_of_nodes, resources = 'omnipath')
```

In order to use a different database for the Universe, it is necessary to provide it directly to the Universe class, using the appropriate functions. As example, we report the steps necessary to integrate Signor and Huri in the Universe class. The same example is reported in the NeKo documentation.

Strategy	Python Function
Reciprocal Pathway Extender	<i>Network.complete_connection</i> and <i>Network.connect_subgroup</i>
Iterative Neighbor Expansion	<i>Network.connect_radially</i>
Regulatory Cascade Explorer	<i>Network.connect_to_upstream_nodes</i>
Module Connection Mapper	<i>Network.connect_component</i>
Phenotype Integration and Network Connectivity	<i>Network.connect_genes_to_phenotype</i>

- Signor

- Download the file from the Signor webpage: <https://signor.uniroma2.it/downloads.php>. Select "Complete Dataset", chose the organism "Human" and select the ".tsv" format.
- Place the downloaded file in your NeKo project folder.
- In a Jupyter notebook, import the NeKo modules:

```
from neko.core.network import Network
from neko.inputs import Universe, signor
```

- Once imported the modules, create the Universe providing the path of the .tsv file downloaded previously. Build the database.

```
resources = Universe()
resources = signor("../neko/_data/signor_db.tsv")
resources.build()
```

- Create the Network object specifying the newly created Universe as resources:

```
neko_network = Network(list_of_nodes, resources = resources.interactions)
```

- Huri

- Download one or more of the files from the Huri webpage: <http://www.interactome-atlas.org/download>. For the use cases shown in the main text, we selected all the files from Luck et al [?] but for this example we will focus on the file "HI-union".
- Once downloaded the file (in .tsv format), move it in your Neko project folder.

- The file contains just two culumns with the Ensamble ids of the proteins. In this case our Universe is unsigned. In order to use this database, we need to translate the identifiers from Ensamble to Uniprot. To do this, we can use the Translator module within NeKo.

- In a Jupyter notebook, import the translator:

```
from neko.inputs.db_translator import IDTranslator
```

- Translate al the identifiers using the following function:

```
translator = IDTranslator('HI-union.tsv', 'genes_translated.csv', 'Ensembl', 'UniProtKB-Swiss-Prot', has_header=False, input_columns=['source', 'target'], processes=12)
translator.run()
```

- The function will start traslating the unique identifiers of the database. This process can take up to one hour, but at each step, a pickle file will be created to save the progress of the translation. In case the translation stops, it can be resumed from the latest saved pickle file. A new file called "gene\_translated.csv" will be created. This file will be used to create a NeKo Universe.

- The file contains just two columns and no information about the sign of the interaction. We can use the function implemented in the Universe class to adapt it to the NeKo structure.
- We start by creating the Universe. Next, we define a dictionary, specifying which column we want to use as "source" and which one we want to use as "target".

```
resources = Universe()
mapping = {"source_UniProtKB-Swiss-Prot": "source", "target_UniProtKB-Swiss-Prot": "target"}
resources.add_resources(huri, columns=mapping, reset_index=True)
```

- Finally we build the Universe and create the Network object specifying the newly created Universe as resources:

```
resources.build()
neko_network = Network(list_of_nodes, resources = resources.interactions)
```

More examples on how to build a custom Universe can be found in the NeKo documentation, in the notebook "Build network using user-defined resources".

## S5 Additional analysis on Use case 1 - differences between Signor and HuRI

An important aspect to take into account when using NeKo, is the impact of the chosen Universe (the database containing the interactions to mine) on the final network topology. Databases like Omnipath, contain a high number of interaction from multiple sources. but this can lead to introduce some off-topic interactions in the network (i.e. all the interactions that are un-related to the initial biological question).

To highlight the differences that arise from using different databases, we have conducted the same analysis proposed in the main text for Use Case 1 (which was based on the HuRI database) starting from the Signor database.

While HuRI focuses on physical protein-protein interactions resulted from experimental techniques, Signor 3.0 focuses on the causal relationships between proteins.

The advantage of using databases like Signor is that the type of interaction is known including the directionality, resulting in directed and signed networks. On the other hand, many interactions in Signor have been reported both as activation and inhibition, depending on the context. In Neko, given a specific context, the user needs to further refine the template model by manually selecting the sign of the "bimodal" interaction. Figure S3 report the networks for each subgroup built from both Signor (directed and signed) and HuRI (directed but not signed). The networks built starting from Signor contains both positive (green arrows) and negative (red arrows) interactions. Moreover, when possible (according to the database annotations), NeKo can recognize complex formation (blue arrows). Given the intrinsic properties of PPI networks, networks inferred on HuRI database are unsigned and undirected.

Figure 4 and S4 show the networks inferred based on hallmark genes characterizing four different subgroups, which were manually selected from Northcott et al. and Lin et al., using Huri and Signor, respectively HuRI and Signor do not contain the same number of nodes. Some genes available for HuRI were not available for Signor. This point introduces an intrinsic bias in the construction of the networks using different databases determining overall, a lower number of edges and nodes in the network created with Signor (Fig. S4 B) and a higher one when using HuRI (Fig. ?? A). The networks built using the hallmark genes of WNT pathway using HuRI and Signor show different topologies. First, the network obtained via HuRI has been built with an input gene list that, after filtering, was bigger than the one used as input for Signor. Both databases can be interpreted as networks following a scale-free distribution. However, given the different topological constraints, (HuRI undirected, Signor directed) HuRI has in general a higher clustering coefficient because of dense modular clusters, while the hierarchical structure of Signor, typical of networks representing signaling pathways, is overall lower. This leads to a different number of edges available for the networks obtained using the different networks (347 edges with Signor and 260 with HuRI; 102 Nodes with Signor and 97 with HuRI). This observation matches with HuRI's

property of "small-world" network diameter that implies a consequent average shorter path between elements of the inferred subnetwork compared to the one inferred with Signor.

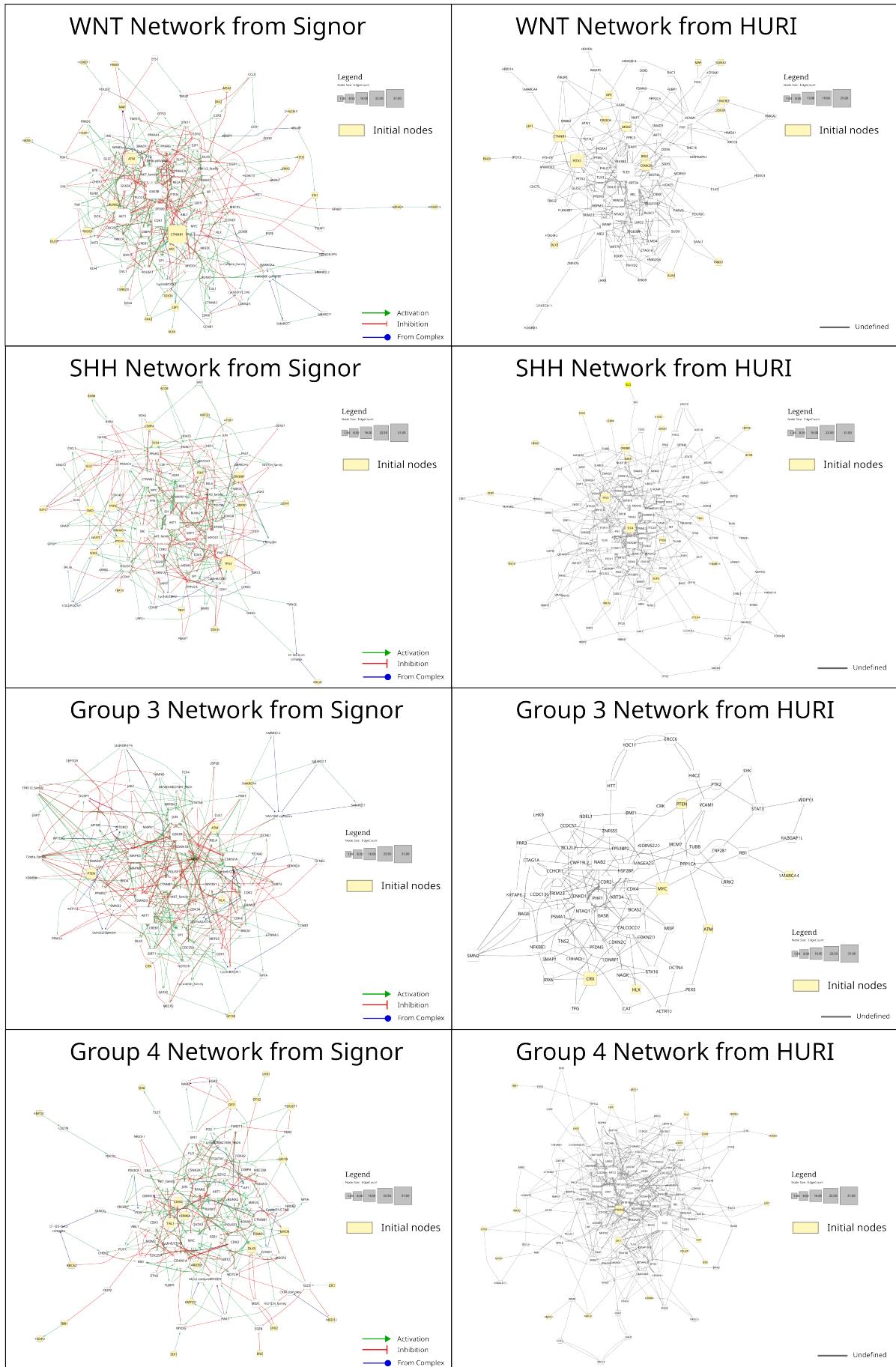
Starting genes for each group

WNT		SHH		Group3		Group4	
ALX4	HOXD13	ATOH1	PRKAR1A	ATM	ARID5B	MED12	
APC	HOXD8	BBX	PTCH1	CRX	BARHL1	MYCN	
ARID1A	HOXD9	BCOR	PTEN	GFI1B	BRCA2	OTX2	
ARID2	IRX2	BRCA2	RARB	HLX	CDK6	POU2F1	
ATM	IRX5	CEBPA	SMO	IRX6	CUX2	PRDM6	
BAI3	LEF1	CREBBP	SOX13	KBTBD4	DLX5	SIX1	
CSNK2B	LHX8	DDX3X	SOX2	LHX9	EN2	SIX6	
CTNNB1	MAF	DMRTA1	SUFU	MYC	EOMES	TAL1	
DDX3X	MSX2	EPHA7	TBX18	NKX3-2	FOXP2	TBR1	
DLX3	NKX6-1	FBXW7	TCF4	PTEN	GFI1	UNCX	
DLX4	ONECUT2	GLI2	TERT	SMARCA4	GFI1B	ZIC1	
EMX2	OSR2	GSE1	TP53		KBTBD4	ZMYM3	
EN1	PAX3	IDH1			KDM6A		
EPHA7	PIK3CA	KMT2C			KMT2C		
FOXF1	PITX1	KMT2D			KMT2D		
HOMEZ	PRRX1	NFATC1			LHX1		
HOXC4	RUNX2	PBX1			LHX2		
HOXC5	SHOX2	POU3F3			LHX5		
HOXD11	SYNCRIP	POU6F2			LMX1A		

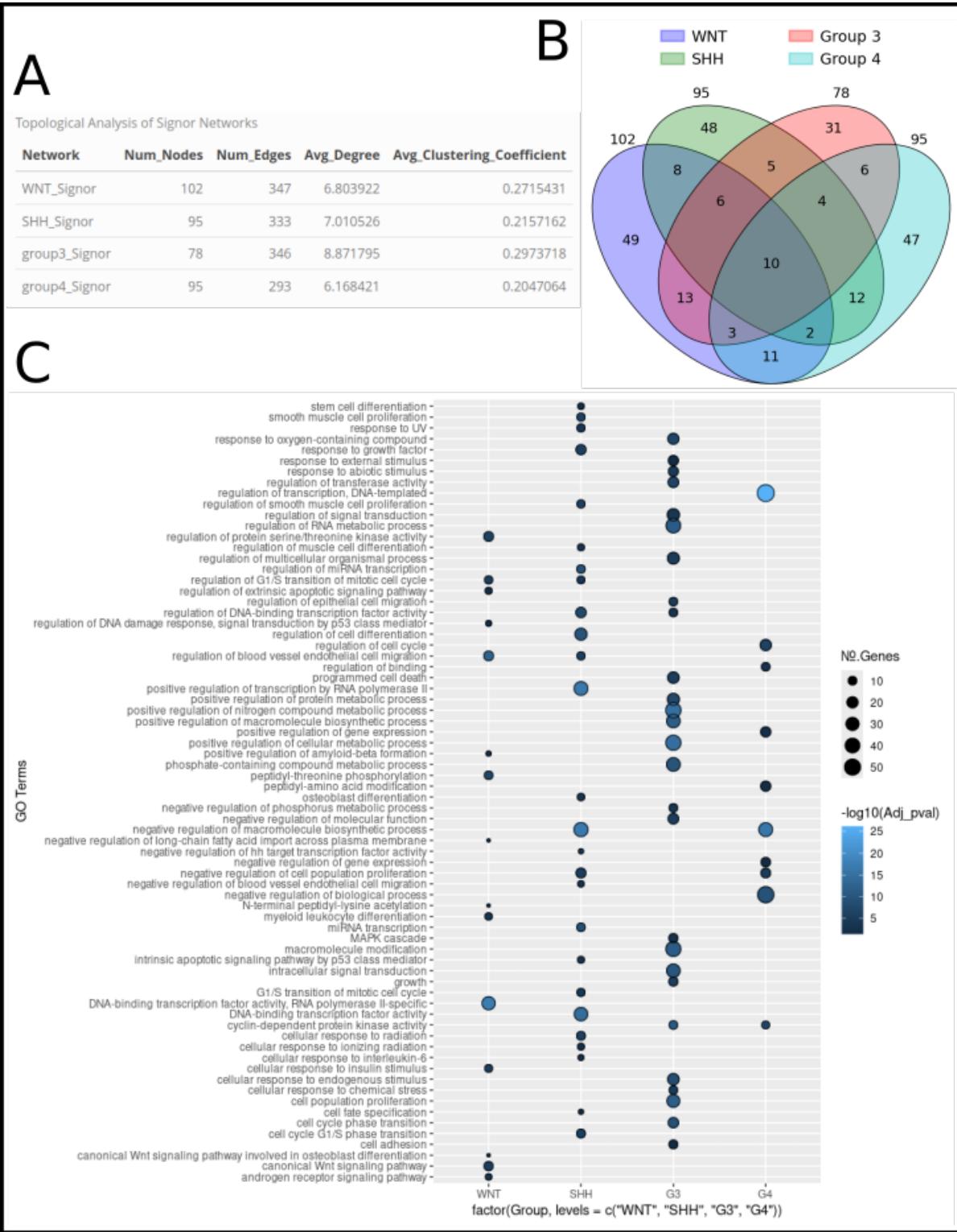
**Legend:**

- Blue: Genes present in both databases
- Purple: Genes present just in Huri
- Green: Genes present just in Signor
- White: Genes not found in any database

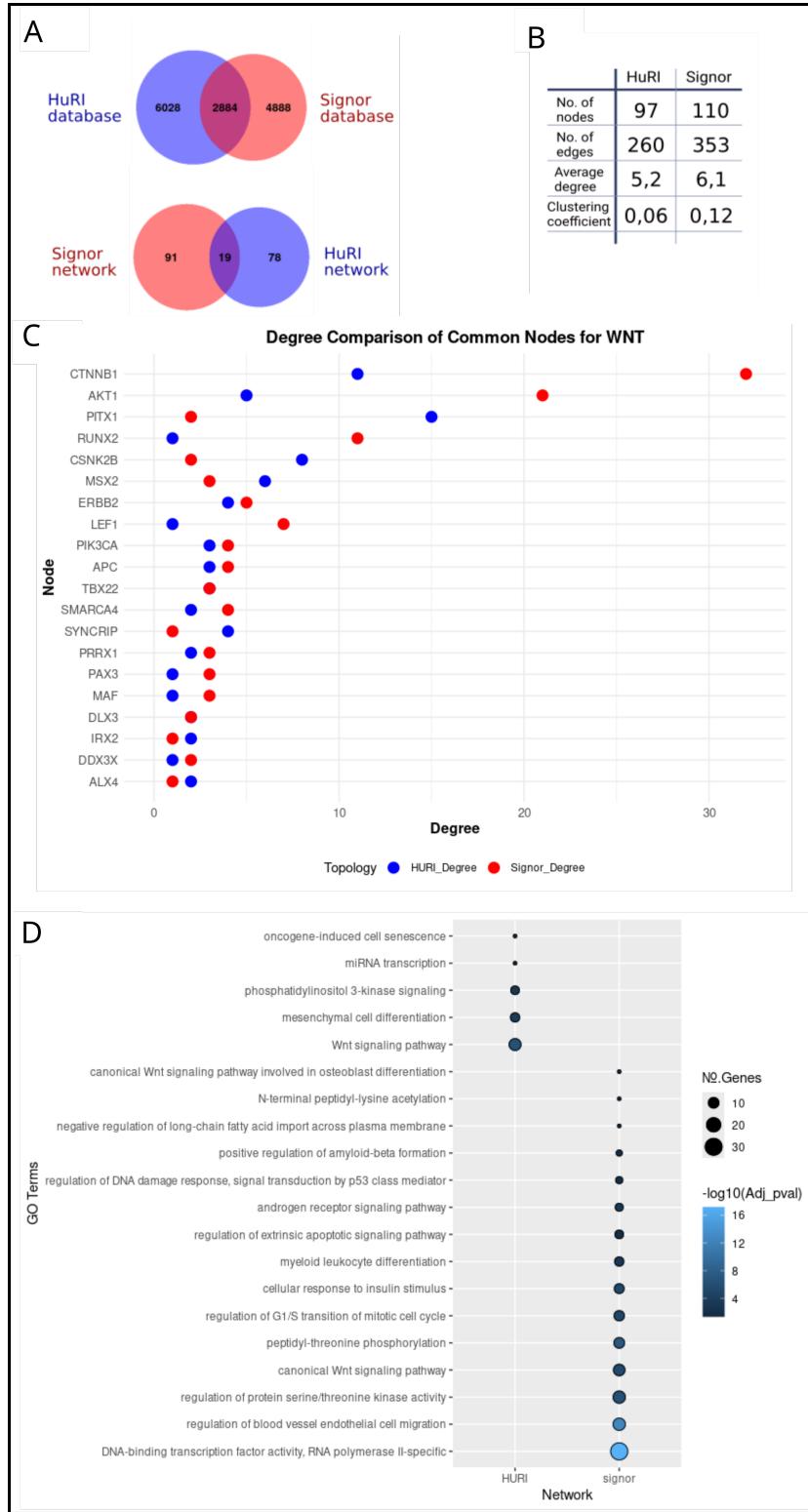
**Figure S2.** Initial genes selected per each group. Not all the selected genes were present in the database or managed to be connected with any paths of length minor or equal 4. The color in the table shows which genes were found in Huri, Signor, both or none of them.



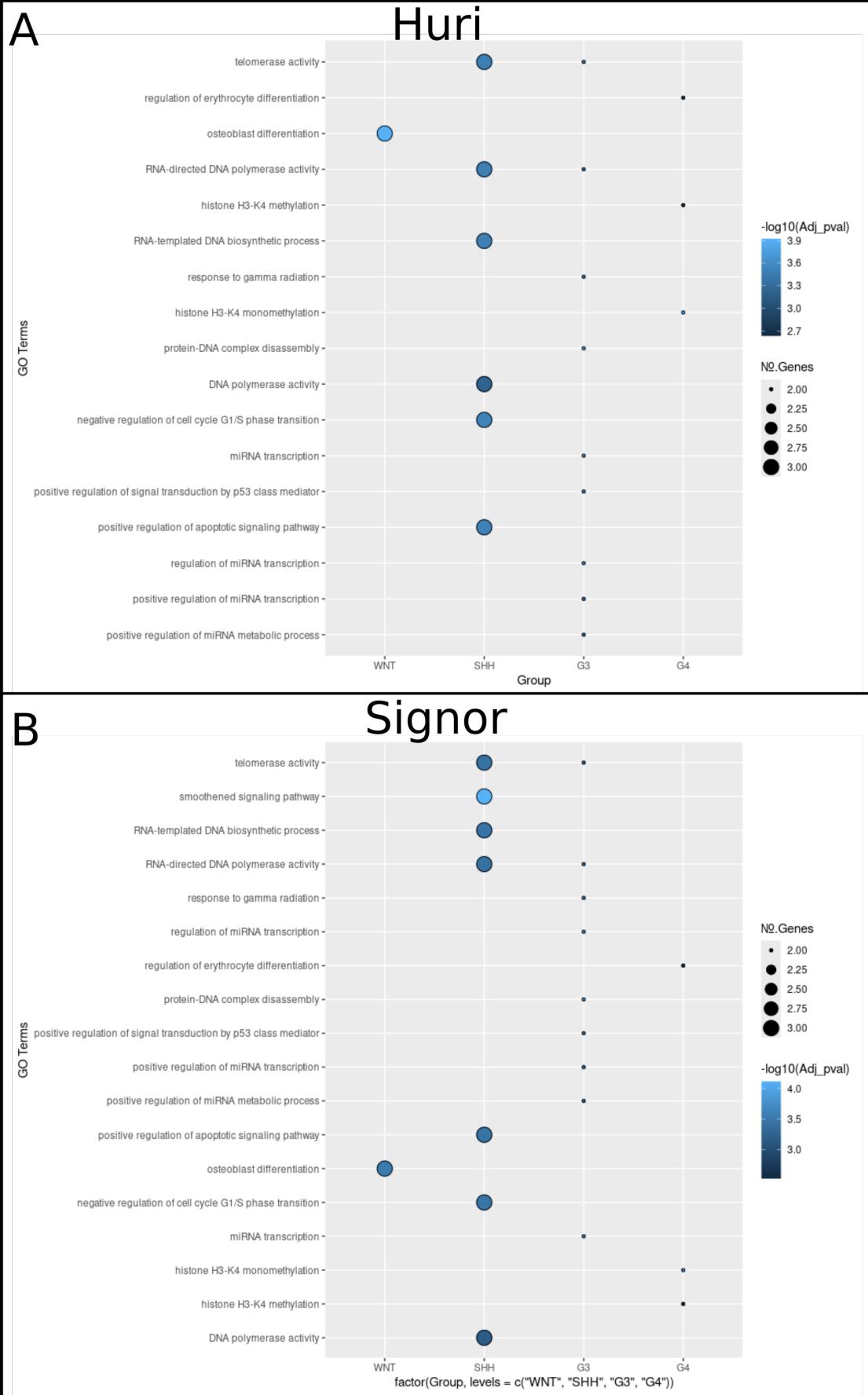
**Figure S3.** Cytoscape visualization of each subgroup's network generated using Signor and HURI.



**Figure S4.** Topological and functional comparison of the NeKo networks created for the medulloblastoma subgroups using the Signor databases. A) Comparison of the main topological networks properties between the four subgroups B) Venn diagram displaying the common nodes among each subgroups. C) Functional comparison of the four medulloblastoma subgroups network generated from the Signor database, using the ClueGO Cytoscape plug-in. Only leading terms with an adjusted p-value  $\leq 0.05$  are presented.

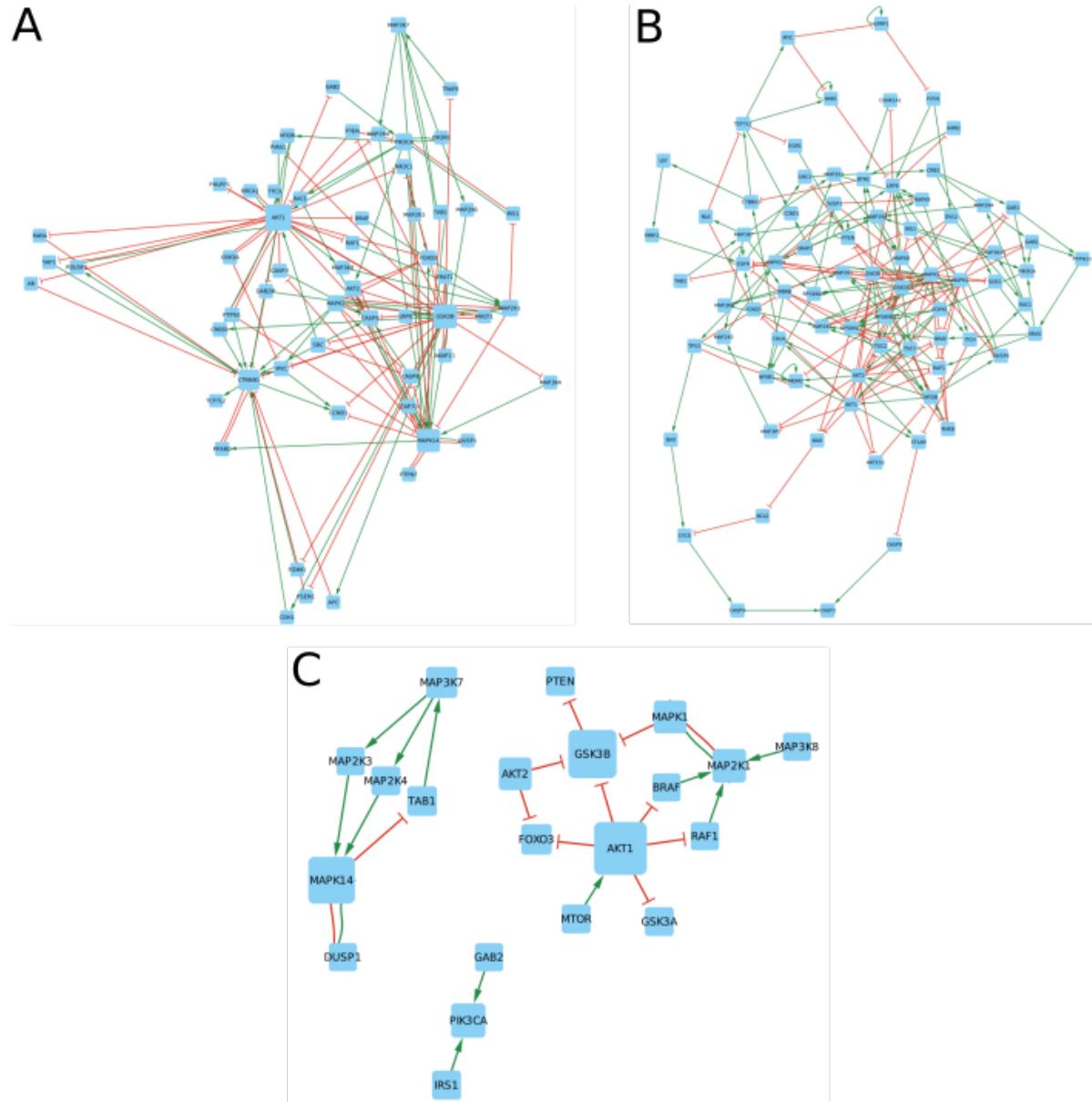


**Figure S5.** Topological and functional comparison of the NeKo networks created for the medulloblastoma subgroups using the Signor database, with a focus on the WNT subgroup. A) Comparison of the total number of nodes included in the HuRI and Signor databases (upper diagram) and the number of nodes included in their respective networks for the WNT subgroup. B) Comparison of the main topological networks properties between the two databases C) Comparison of the degree of the common nodes of the WNT subgroup networks built from HuRI and Signor. D) GO term overrepresentation comparison between the WNT subgroup networks built from Signor and HuRI.



**Figure S6.** GO term overrepresentation of the seeds genes present in A) the Huri database and B) the Signor 3 database.

## S6 Additional analysis on Use case 2 - Topological differences between Flobak et al. and NeKo



**Figure S7.** A) Cytoscape visualization of the network obtained with NeKo B) Cytoscape visualization of the cascade network, C) common interactions between the two networks.

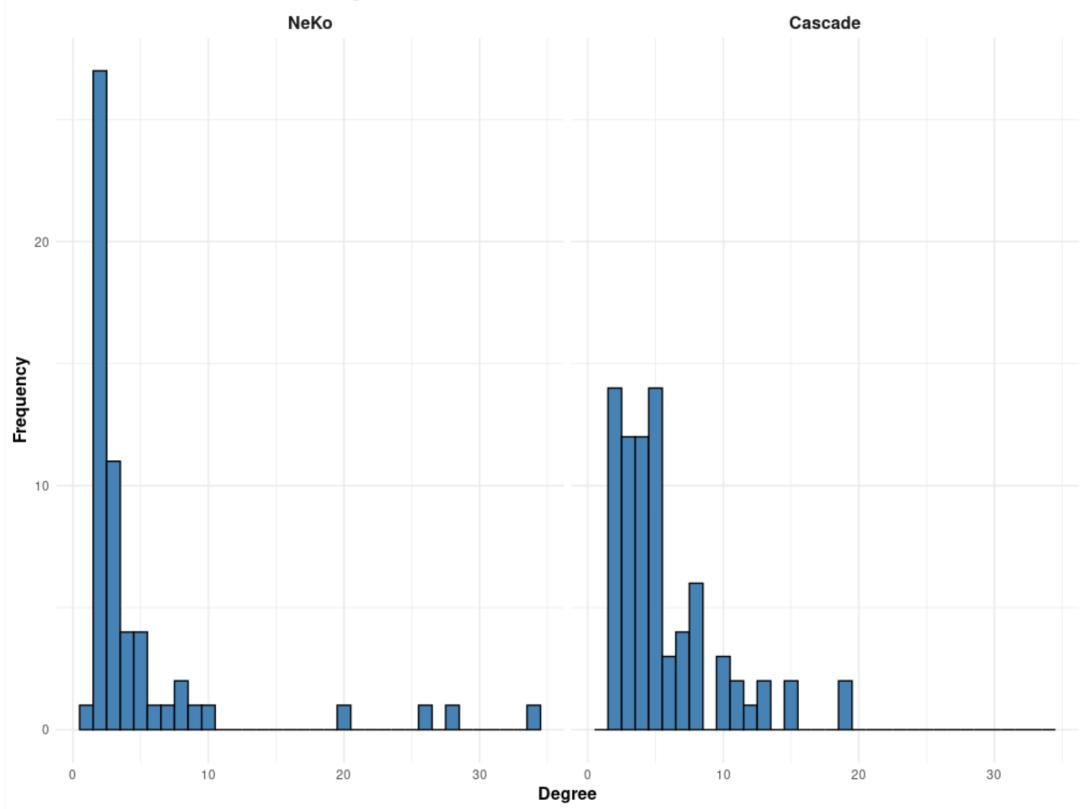
**A**

## Topological Analysis of NeKo and Cascade Networks

Network	Num_Nodes	Num_Edges	Avg_Degree	Avg_Clustering_Coefficient
NeKo	57	140	4.912281	0.4283600
Cascade	77	217	5.636364	0.0681066

**B**

Degree Distribution for NeKo and Cascade Networks

**Figure S8.** A) Topological overview of the two networks, B) Degree distribution.